fq
jq for binary formats
Mattias Wadman
What is fq?

- Tool, language and decoders for working with data
- Inspired and based on jq
- Query and display data in a useful ways
- Interactive REPL with auto-completion
- Available for Linux, macOS, Windows, BSD
- Debugger for files
Why jq?

- CLI friendly syntax
  - Terse and composable: `a | b | ...
  - Generators to iterate and recurse: `.[] | ..., .. | ...

- DSL to select and transform data
  - Superset of JSON: `{a: [1,2+3,empty]} ➞ "a": [1,5]`
  - `{"a": 1, "b": 2} ➞ {sum: (.a+.b)} ➞ "sum": 3`

- Purely functional language based on generators and backtracking
  - Conditionals, functions, bindings, special forms to collect and reduce etc

- Single filter run on each input
  - Can behave different using inputs, slurp, etc
Supported formats

- Currently 113 formats
- Media containers (MP3, MP4, Ogg, FLAC, Matroska, PNG, JPEG, Exif, ...)
  - Some demux and decode samples (AAC, NALU, h264, MP3 frames, ...)
- Executables (ELF, Macho, Wasm)
- Archives and compression (ZIP, Tar, gz, bz2, ...)
- Network protocols (PCAP, Ethernet, UDP, TCP, IPV4, ...)
  - TCP reassembly and IPv4 defragmentation
- Serialization formats (MsgPack, ASN1 BER, CBOR, ProtoBuf, bplist, ...)
- Can also decode/encode text formats (YAML, XML, HTML, Toml, CSV, URLs, ...)
Decode in what way?

- A format decoder produces a JSON compatible structure
  - Each value has a bit range and optional description, mapping etc
- For media usually decode most things except the actual media
- For other formats more or less everything
- Some formats broken down into independent sub formats
  - `flac` into `flac_frame`, `flac_metadatablock`, `flac_streaminfo`, ...
- Format can use other formats
  - `pcap` uses `ether8023_frame` that uses `ipv4_packet` that uses `tcp_segment`
  - `mp3 → id3v2 → jpeg → exif → icc_profile`
  - Can pass data between formats
My own use cases for fq

SIMPLY EXPLAINED:
METADATA

WHAT'S "METADATA"?

A WORD WITH 8 LETTERS

My own use cases for fq

“multimedia is basically neverending pain”
— wm4, mpv-player lead

- That file actually has an end
  - Segment linking or `dref` atom loop (A → B → A → B → …)
  - LOOP entry in udta atom
- That the video track is even a video “codec”
  - PowerPoint-in-MOV, FIRE, VNC commands, RAR in MKV
- That you can easily downmix or play a stereo track on a mono speaker
  - Phase cancellation is great fun

Derek Buitenhuis - Things Developers Believe About Video Files (Proven Wrong by User Uploads) (https://www.youtube.com/watch?v=cR5D3RU00A)
My own use cases for fq

- Debug, query and assist when working with media files
- Show overview to look for unusual value or structures
- Sample tables, timestamps, decoder configuration, ...
- Find broken media sample
- Automate and aggregate over multiple files
What it can't do

- Very little encoding support, focus on decoding
  - Not clear how it would work (`mov.c` + `movenc.c` in ffmpeg ~17000 lines of dense C)
  - Tip! See next talks by Petr and Jose, both will touch on this I think.
- Can decode broken things but can't magically repair
  - Makes it possible but you probably need deep knowledge
- Decoders in runtime
  - Want to look into decoders in jq
  - Something declarative like Kaitai struct would be nice
  - Currently can slice and construct new binaries
Why go?

- Previous experience with go
- Two mature jq implementation
  - Original jq in C
  - gojq in go
- Robust, complex and safe decoders in C? 😐
  - Probably need a VM etc
- Great tooling, fast builds, static binaries, cross platform
def input:
    (next_filename
     | open
     | decode)

def inputs: repeat(input);

def main:
    (parse_arguments as $opts
     | setup_some_global_state
     | if $opts.repl then repl else
       (inputs
        | eval($opts.filter)
        | display)
    )
end
func decodeFoss(d *decode.D, in any) any {
    var length uint64
    d.FieldStruct("header", func(d *decode.D) {
        d.FieldUTF8("signature", 4, d.StrAssert("foss"))
        d.FieldU8("license", scalar.UintMapSymStr{
            0: "GPL",
            1: "MIT",
        })
        length = d.FieldU16("length")
    })
    d.FramedFn(int64(length)*8, func(d *decode.D) {
        d.FieldArray("sections", func(d *decode.D) {
            for !d.End() {
                d.FieldFormat("section", sectionFormat, nil)
            }
        })
    })
    return nil
}
```
$ fg . file.foss

<table>
<thead>
<tr>
<th>0x00</th>
<th>00 01 02 03 04 05 06 07</th>
<th>01234567</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>66 6f 73 73 01 00 0d</td>
<td>foss...</td>
</tr>
<tr>
<td>0x00</td>
<td>05</td>
<td>.</td>
</tr>
<tr>
<td>0x08</td>
<td>68 65 6c 6c 6f 06 66 6f</td>
<td>hello.fo</td>
</tr>
<tr>
<td>0x10</td>
<td>73 64 65 6d</td>
<td>sdm</td>
</tr>
</tbody>
</table>

$ fg d file.foss

<table>
<thead>
<tr>
<th>0x00</th>
<th>00 01 02 03 04 05 06 07</th>
<th>01234567</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>66 6f 73 73</td>
<td>foss</td>
</tr>
<tr>
<td>0x00</td>
<td>01</td>
<td>.</td>
</tr>
<tr>
<td>0x00</td>
<td>00 0d</td>
<td>..</td>
</tr>
<tr>
<td>0x00</td>
<td>00 01 02 03 04 05 06 07</td>
<td>01234567</td>
</tr>
<tr>
<td>0x00</td>
<td>05</td>
<td>.</td>
</tr>
<tr>
<td>0x08</td>
<td>68 65 6c 6c 6f</td>
<td>hello</td>
</tr>
<tr>
<td>0x08</td>
<td>00 01 02 03 04 05 06 07</td>
<td>01234567</td>
</tr>
<tr>
<td>0x08</td>
<td>06</td>
<td>.</td>
</tr>
<tr>
<td>0x08</td>
<td>66 6f</td>
<td>fo</td>
</tr>
<tr>
<td>0x10</td>
<td>73 64 65 6d</td>
<td>sdm</td>
</tr>
</tbody>
</table>

$ fg ".sections[1]" file.foss

<table>
<thead>
<tr>
<th>0x08</th>
<th>00 01 02 03 04 05 06 07</th>
<th>01234567</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x08</td>
<td>06</td>
<td>.</td>
</tr>
<tr>
<td>0x08</td>
<td>66 6f</td>
<td>fo</td>
</tr>
<tr>
<td>0x10</td>
<td>73 64 65 6d</td>
<td>sdm</td>
</tr>
</tbody>
</table>
```

```json
$ fq tovalue file
{
  "header": {
    "length": 13,
    "license": "MIT",
    "signature": "foss"
  },
  "sections": [
    {
      "length": 5,
      "text": "hello"
    },
    {
      "length": 6,
      "text": "fosdem"
    }
  ]
}

$ fq torepr file
{
  "sections": [
    "hello",
    "fosdem"
  ],
  "type": null
}
```
hello fosdem
Thanks and useful tools

- itchyny for gojq
- Stephen Dolan and others for jq
- HexFiend
- GNU poke
- Kaitai struct
- Wireshark
- vscode-jq (https://github.com/wader/vscode-jq)
- jq-lsp (https://github.com/wader/jq-lsp)
Thank you

jq for binary formats

Mattias Wadman
mattias.wadman@gmail.com (mailto:mattias.wadman@gmail.com)
https://fosstodon.org/@wader (https://fosstodon.org/@wader)